On the complexity of freezing automata networks of bounded pathwidth*

Eric Goles[1], Pedro Montealegre[1], Martín Ríos-Wilson[1], and G. Theyssier[2]

[1] Facultad de Ingeniería y Ciencias, Universidad Adolfo Ibáñez, Santiago
[2] I2M (Aix-Marseille University, CNRS)

Abstract. An automata network is a graph of entities, each holding a state from a finite set and evolving according to a local update rule which depends only on its neighbors in the network's graph. It is freezing if there is an order on the states such that the state evolution of any node is non-decreasing in any orbit. They are commonly used to model epidemic propagation, diffusion phenomena like bootstrap percolation or cristal growth.
Previous works have established that, under the hypothesis that the network graph is of bounded treewidth, many problems that can be captured by trace specifications at individual nodes admit efficient algorithms. In this paper we study the even more restricted case of a network of bounded pathwidth and show two hardness results that somehow illustrate the complexity of freezing dynamics under such a strong graph constraint. First, we show that the trace specification checking problem is NL-complete. Second, we show that deciding first order properties of the orbits augmented with a reachability predicate is NP-hard.

1 Introduction

Automata networks (AN) are finite dynamical systems that can be seen as the finite and non-uniform counterpart of cellular automata on arbitrary graphs. An automata network is *freezing* if there is an order on the states such that the state evolution of any node is non-decreasing in any orbit. Several models that received a lot of attention in the literature are actually freezing automata networks, for instance: bootstrap percolation which has been studied on various graphs [1, 4, 3, 11], epidemic [6] or forest fire [2] propagation models, cristal growth models [17, 10] and various models of self-assembly tilings [18].

The freezing condition has strong implications on the computational complexity of these systems. For instance, following previous works on cellular automata [14, 8], it was established in [9] that a large set of problems specified by traces at individual nodes are actually NC when considering freezing automata networks of bounded treewidth. This result in particular captures the problem

* Research partially supported by projects STIUC-AMSUD 22-STIC-02 (all authors), Fondecyt-ANID 1200006 (EG), FONDECYT-ANID 1230599 (PM), ANID FONDECYT Postdoctorado 3220205 (MRW)

of nilpotency, a property which can be expressed in the language of orbits by: all orbits converge to the same fixed point. The nilpotency problem is typical of the computational complexity collapse when the freezing condition is combined by a condition on the structure of the network.

This paper aims at better understanding this complexity collapse by giving lower bounds for freezing automata networks on the simplest network structure: graphs of *bounded pathwidth* (intuitively, that are structurally close to a line or a cycle).

First, we consider regular trace properties (*i.e.* regular expressions specifying allowed traces at each node) and show that the problem of existence of an orbit following the constraints is NL-complete (Theorem 2). Note that this problem is similar to some well-studied problems in 1D cellular automata like cylinder-to-cylinder reachability which can also be expressed as a regular expression of traces. It is striking to compare the finite context with the NL upper bound above to the infinite context, where freezing cellular automata have actually an undecidable cylinder-to-cylinder reachability problem [14].

Second, we study another family of problems : properties defined by first order logic on configuration with equality, a predicate $x \to y$ meaning that y can be reached from x in one step, and a predicate $x \to^+ y$ meaning that configuration y can be reached from configuration x in some number of steps. This logic denoted FO$^+$also captures nilpotency by $\exists! y, \forall x : x \to^+ y$. Our second main result is that, although nilpotency is co-NL (Corollary 1), the model checking of FO$^+$is NP-hard even for freezing automata networks defined on a line (Theorem 3).

Our results contribute to the following global picture where each cell of the table is divided between the general case (lower left in black) and the freezing case (upper right in blue).

	Infinite 1D CA	Finite bounded pathwidth AN
Nilpotency	Decidable [14, Theorem 2] / Undecidable [12]	co-NL (Corollary 1) / PSPACE-complete [7, Corollary 3.2]
Regular trace properties	Undecidable [14, Theorem 5] / Undecidable	NL-complete (Theorem 2) / PSPACE-complete [7, Theorem 3.3]
FO$^+$	Open / Undecidable [12]	NP-hard (Theorem 3) / PSPACE-complete [7, Corollary 3.2]

2 Definitions and notations

Given a graph $G = (V, E)$ and a vertex v we will call $N(v)$ the neighborhood of v and δ_v to the degree of v. In addition, we define the closed neighborhood of v

as the set $N[v] = N(v) \cup \{v\}$ and we use the following notation $\Delta(G) = \max_{v \in V} \delta_v$ for the maximum degree of G. We will use the letter n to denote the order of G, i.e. $n = |V|$. Also, if G is a graph and the set of vertices and edges is not specified we use the notation $V(G)$ and $E(G)$ for the set of vertices and the set of edges of G respectively. In addition, we will assume that if $G = (V, E)$ is a graph then, there exists an ordering of the vertices in V from 1 to n. During the rest of the text, every graph G will be assumed to be connected and undirected. We define a *class* or a *family* of graphs as a set $\mathcal{G} = \{G_n\}_{n \geq 1}$ such that $G_n = (V_n, E_n)$ is a graph and $|V_n| = n$.

Non-deterministic freezing automata networks. Let Q be a finite set that we will call an *alphabet*. We define a non-deterministic automata network in the alphabet Q as a tuple $(G = (V, E), \mathcal{F} = \{F_v : Q^{N(v)} \to \mathcal{P}(Q) | v \in V\}))$ where $\mathcal{P}(Q)$ is the power set of Q. To every non-deterministic automata network we can associate a non-deterministic dynamics given by the global function $F : Q^n \to \mathcal{P}(Q^n)$ defined by $F(x) = \{x \in Q^n | x_v \in F_v(x), \forall v\}$.

Definition 1. *Given a a non-deterministic automata network $(G, \mathcal{F})$ we define an orbit of a configuration $x \in Q^n$ at time t as a sequence $(x_s)_{0 \leq s \leq t}$ such that $x_0 = x$ and $x_s \in F(x_{s-1})$. In addition, we call the set of all possible orbits at time t for a configuration x as $\mathcal{O}(x, t)$. Finally, we also define the set of all possible orbits at time t as $\mathcal{O}(\mathcal{A}, t) = \bigcup_{x \in Q^n} \mathcal{O}(x, t)$*

We say that a non-deterministic automata network $(G, \mathcal{F})$ defined in the alphabet Q satisfies the *freezing property* or simply that it is *freezing* if there exists a partial order $\leq$ in Q such that for every $t \in \mathbb{N}$ and for every orbit $y = (x_s)_{0 \leq s \leq t} \in \mathcal{O}(\mathcal{A}, t)$ we have that $x_s^i \leq x_{s+1}^i$ for every $0 \leq s \leq t$ and for every $0 \leq i \leq n$.

Path decompositions and pathwidth. Let $G = (V, E)$ be a connected graph. A subgraph P of G is said to be a path if $V(P) = \{v_1, \ldots, v_k\}$ where every v_i is different and $E(P) = \{v_1v_2, v_2v_3 \ldots, v_{k-1}v_k\}$. Now we present a graph parameter called *pathwidth* which, generally speaking, indicates how similar a graph is to a path graph. More precisely, we have the following definition:

Definition 2. *Given a graph $G = (V, E)$ a path decomposition is pair $\mathcal{D} = (P, \Lambda)$ such that P is a path graph and Λ is a family of subsets of nodes $\Lambda = \{X_t \subseteq V |\ t \in V(T) = \{1, \ldots, s\}\}$, called bags, such that:*

- *Every node in G is in some X_t, i.e: $\bigcup_{t \in V(P)} X_t = V$,*
- *For every $e = uv \in E$ there exists $t \in V(P)$ such that $u, v \in X_t$,*
- *For every $u, v, w \in V(P)$ if $1 \leq u < v < w \leq s$ then, $X_u \cap X_v \subseteq X_w$.*

We define the width of a path decompostion $\mathcal{D}$ as the amount $\text{width}(\mathcal{D}) = \max_{t \in V(P)} |X_t| - 1$. Given a graph $G = (V, E)$, we define its pathwidth as the parameter $\text{path}(G) = \min_{\mathcal{D}} \text{width}(\mathcal{D})$. In other words, the pathwidth is the minimum width of a path decomposition of G. Note that, if G is a connected graph such that $|E(G)| \geq 2$ then, G is a path if and only if $\text{path}(G) = 1$.

It is known that a path decomposition can be computed in **DLOGSPACE** [13].

Specification checking problem. Now, we introduce a decision problem called *specification checking problem.* Roughly, this problem ask for the existence of an orbit in the automata network that verifies some trace constraints at each node. The information of allowed traces at each node is called a *specification*: a specification of length t is a map $\mathcal{E}_t : V \to \mathcal{P}(Q^t)$ such that, for every $v \in V$, the sequences in $\mathcal{E}_t(v)$ are non-decreasing (and thus respect the freezing condition). We say that $\mathcal{E}_t$ is satisfiable by $\mathcal{A}$ if there exists an orbit $O \in \mathcal{O}(\mathcal{A}, t)$ such that $O_v \in \mathcal{E}_t(v)$ for every $v \in V$. We observe that the number of freezing traces of length t is polynomial in t so $\mathcal{E}_t$ can be represented in polynomial size in V and t.

Also, in the absence of explicit mention, all the considered graphs will have bounded degree Δ by default, so a freezing automata network rule can be represented as the list of local update rules for each node which are maps of the form $Q^\Delta \to \mathcal{P}(Q)$ whose representation as transition table is of size $O\big(|Q|^{\Delta+1}\big)$. The specification checking problem (SPEC) introduced in [9] asks whether a given freezing automata network satisfies a given specification. If $\mathcal{E}_t$ is a satisfiable t-specification for some automata network $\mathcal{A}$ we write $\mathcal{A} \models \mathcal{E}_t$.

In [9] it is shown that many well-known and well-studied decision problems related to the dynamics of automata networks are somehow related to SPEC. These problems are: the prediction problem, the predecessor problem, the nilpotency problem and the asynchronous reachability problem. Recall that nilpotency is the property that there is a configuration x such that all orbits end up in x and x is a fixed point. Most of these problems are sub-problems of SPEC. In the case of Nilpotency, an efficient parallel Turing reduction can be constructed [9].

In this paper, we focus on a variant of the specification problem were admissible traces are represented as regular expressions. More precisely, a regular (Q, V)-specification is a map from V to regular expressions over alphabet Q. We therefore consider the Regular Specification Checking Problem or simply REGSPEC which is the same as SPEC except that the specification must be a regular specification. It is interesting to observe that REGSPEC with fixed degree and fixed treewidth and with alphabet as unique parameter is $W[2]$-hard [9].

3 NL-completeness of REGSPEC problem

In this section, we explore different results for the complexity of REGSPEC when the pathwidth of the underlying interaction graph is bounded. We start this section by showing that REGSPEC is in **NL**. This is a direct extension of the results on bounded treewidth in [9] and the technique used in [14] for the prediction problem in one dimensional freezing cellular automata. Then, we show that the problem is actually **NL**-complete by showing a logspace reduction from (s, t)-connectivity.

Theorem 1. *The REGSPEC problem is in **NL** for bounded pathwidth freezing AN.*

The complement of the nilpotency problem can be reduced to instances of REGSPEC in such a way that we keep the strong complexity upper-bounds from the previous theorem.

Corollary 1. *The nilpotency problem is in* ***co-NL*** *for bounded pathwidth freezing AN.*

Now we show show that REGSPEC is **NL**-complete and thus, it is most likely that the previous algorithm is the best we can do, unless **NL** = **DLOGSPACE**.

Now we introduce the main result of the section.

Theorem 2. *The Regular Specification Checking problem (REGSPEC) is* ***NL****-complete when restricted to bounded degree and bounded pathwidth interaction graphs.*

The proof of this result is divided in what we call *phases.* The main idea of the proof is to construct a non-deterministic automata network $\mathcal{A}_D = (G_D, \mathcal{F}_D)$ defined over a two dimensional grid of size $k \times d$ where $d = n^{\mathcal{O}(1)}$ and $k = \mathcal{O}(1)$. Of course, since k is constant, then $\mathcal{A}_D$ has bounded pathwidth. This automata network will non-deterministically guess a sequences of blocks (a structure representing edges in the interaction graph of $\mathcal{A}$, see Figure 1 for more details). We call this part the *selection phase.* Then, the next part of the proof consists in showing that $\mathcal{A}_D = (G_D, \mathcal{F}_D)$ is capable of deterministically verifying if an initial condition corresponds to a sequence of valid edges, i.e. if it corresponds to a sequence of blocks and they actually represent edges in G_D. We call this phase a *verification phase* In order to perform this task, we use a construction based on using signals that will collide at specific locations as a way to verify the distance between two given cells. In addition, it would be essential to save (as a constant layer) the information contained in the incidence matrix of D. Generally speaking, once $\mathcal{A}_D$ has verified that the sequence of blocks is valid, it will compare two subsequent blocks (which represent a pair of edges) in order to verify if they are incident. If in any part of its dynamics $\mathcal{A}_D$ locally detects some error (by the application of its local rule), it will spread an error state that will led the system to an attractor corresponding to a uniform configuration in which any cell will be in this particular error state. However, if the process runs flawless, then the system will reach an attractor in which all the cells are in a particular success state. We will code, by using a specification $\mathcal{E}_D$ (given in the input of REGSPEC), a specific requirement for the initial configuration (more precisely, we will ask the initial configuration to have the incidence matrix of D, markers and information about the nodes (s, t)) in order to allow $\mathcal{A}_D$ to have enough information to start the selection and verification process. In addition, we will code in this specification only the orbits that will reach this specific success state. By doing this, we will show that $\mathcal{A}_D \models \mathcal{E}_D$ if and only if there is a path between s, t in D. Thus, the reduction will consist on constructing $(\mathcal{A}_D, \mathcal{E}_D)$ from (D, s, t) in **DLOGSPACE**.

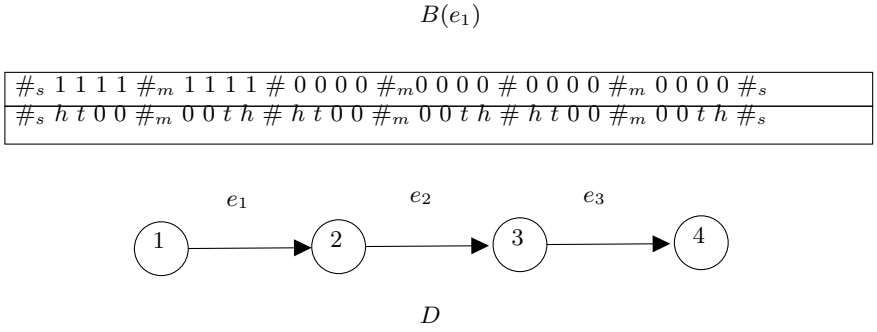


Fig. 1. An example of a block for some graph D.

4 Hardness of FO^+ model checking

$FO(=, \to)$ denotes the first order logic over configurations using equality and a predicate $x \to y$ meaning that y can be reached from x in one step. It is well-known that this logic can be efficiently dealt with using finite automata theory when configurations are one-dimensional. For instance the model-checking of this logic is decidable on one-dimensional CAs [5, 16]. In this subsection, we study first-order properties of the dynamics enriched with a new predicate $x \to^+ y$ expressing that configuration y can be reached from configuration x in some unknown number of steps. We denote this logic $FO^+ = FO(=, \to, \to^+)$. Adding the predicate $\to^+$ allows to express properties like nilpotency:

$$\exists x, \forall z, (z \to^+ z) \implies z = x,$$

which is equivalent in the deterministic case to $\exists x, \forall y, y \to^+ x$. The model checking of FO^+ is therefore undecidable for general 1D CA [12] and **PSPACE-complete** for AN of bounded pathwidth [7]. However, nilpotency is a decidable property for 1D freezing CA [14] and **co-NL** for bounded pathwidth freezing AN (Corollary 1). It is therefore interesting to figure out what is the complexity of the model checking of FO^+ for freezing AN of bounded pathwidth.

The goal of this section is to show that despite considering only "one-dimensional" networks and having the freezing constraint, we can encode bi-dimensional domino problems in FO^+ and thus get a **NP-hard** lower bound. The precise NP-hard problem we consider in this subsection to reduce from is the following.

Lemma 1 (HV-domino CSP). *Let Q be a large enough alphabet. The following problem is NP-complete:*

- ***input:*** *for each $1 \leq i, j \leq n$, two lists of constraints: $H_{i,j} \subseteq Q^2$ and $V_{i,j} \subseteq Q^2$.*
- ***question:*** *does there exist a configuration $a \in Q^{\{1,\dots,n\}\times\{1,\dots,n\}}$ such that for all $1 \leq i, j \leq n$ the local constraints are satisfied, i.e.*

$$(a_{i,j}, a_{i+1,j}) \in H_{i,j} \text{ (if } i < n) \text{ and } (a_{i,j}, a_{i,j+1}) \in V_{i,j} \text{ (if } j < n).$$

We can now show a lower bound on the model checking of a single formula of FO^+. Let $P_x(x)$ denote the formula of FO^+ expressing that configuration x has at least k preimages, formally:

$$P_k(x) \equiv \exists x_1 \neq x_2 \neq \cdots \neq x_k, \bigwedge_{1 \leq i \leq k} x_i \to x.$$

We will consider the following formula ϕ:

$$\phi \equiv \exists x : x \to x$$
$$\wedge \big(\forall y, \forall y^1, \forall z, (\neg P_1(y) \wedge \neg P_2(y^1) \wedge y \to y^1 \wedge y^1 \to^+ z \wedge z \to x \wedge z \neq x) \Rightarrow \neg P_2(z)\big).$$

It expresses that there exists a fixed point x such that considering any orbit starting from a configuration y without preimage, which is the unique preimage of its successor y^1, and leading to x, then the configuration z occurring in the orbit just before reaching x has only 1 preimage.

The main result of this section is that the FO^+ model checking problem is already hard for formula ϕ. The proof uses the HV-domino CSP of Lemma 1. For each HV-domino CSP problem, we build a deterministic automata one-dimensional network that essentially checks that a configuration $(a_{i,j})$ satisfies the HV-domino constraints. By one-dimensional we mean a graph which is a line with self-loops on each node. In this automata network, configurations $(a_{i,j})$ are layed out as one-dimensional configurations so that $a_{i,j}$ and $a_{i+1,j}$ are neighbors in the graph, and therefore H-constraints can be checked locally. However, $a_{i,j}$ and $a_{i,j+1}$ are far away in the graph, so V-constraints require the dynamics of the automata network to be checked. The key idea is to use formula ϕ above to characterize the part of the dynamics of the automata network that checks all V-constraints for a given candidate configuration $(a_{i,j})$: intuitively, quantifying over orbits starting from a configuration y without preimage and being the unique preimage of its successor ensures that the orbit contains some well-initialized computation, and predicate $\neg P_2$ on the configuration before reaching the fixed point codes the fact that the output of the computation is correct. The fixed point configuration x in formula ϕ represents a candidate configuration $(a_{i,j})$ (cleaned from any trace of computation) and, by construction of the automata network, the second part of the formula expresses that for any well-initialized test of a V-constraint the output of the test is a success. Formula ϕ uses predicate $\neg P_2$ to characterize some specific configurations: the key corresponding trick in the construction below is to make Cartesian products of some alphabet with $\{0, 1\}$ and ensure that the action of the automata network almost always reset to 1 the value of such a $\{0, 1\}$-component in at least one node. This ensures that the configuration obtained after one step has at least two preimages. The situations where it is not the case are exceptional and well-controlled: this helps to identify possible candidates for configurations y, y^1 and z in formula ϕ.

Theorem 3. *Checking whether a given deterministic freezing automata network $(G, \mathcal{F})$ verifies ϕ is NP-hard, even when restricted to bounded alphabet, and degree 3 and pathwidth 1.*

References

1. Hamed Amini and Nikolaos Fountoulakis. Bootstrap percolation in power-law random graphs. *Journal of Statistical Physics*, 155(1):72–92, feb 2014.
2. Per Bak, Kan Chen, and Chao Tang. A forest-fire model and some thoughts on turbulence. *Physics Letters A*, 147(5):297 – 300, 1990.
3. József Balogh and Béla Bollobás. Bootstrap percolation on the hypercube. *Probability Theory and Related Fields*, 134(4):624–648, jul 2005.
4. József Balogh, Béla Bollobás, Hugo Duminil-Copin, and Robert Morris. The sharp threshold for bootstrap percolation in all dimensions. *Transactions of the American Mathematical Society*, 364(5):2667–2701, may 2012.
5. Olivier Finkel. On decidability properties of one-dimensional cellular automata. *J. Cellular Automata*, 6(2-3):181–193, 2011.
6. M.A. Fuentes and M.N. Kuperman. Cellular automata and epidemiological models with spatial dependence. *Physica A: Statistical Mechanics and its Applications*, 267(3–4):471 – 486, 1999.
7. Guilhem Gamard, Pierre Guillon, Kevin Perrot, and Guillaume Theyssier. Rice-like theorems for automata networks. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
8. E. Goles, N. Ollinger, and G. Theyssier. Introducing freezing cellular automata. In *Exploratory Papers of Cellular Automata and Discrete Complex Systems (AUTOMATA 2015)*, pages 65–73, 2015.
9. Eric Goles, Pedro Montealegre, Martín Ríos Wilson, and Guillaume Theyssier. On the impact of treewidth in the computational complexity of freezing dynamics. In Liesbeth De Mol, Andreas Weiermann, Florin Manea, and David Fernández-Duque, editors, *Connecting with Computability*, pages 260–272, Cham, 2021. Springer International Publishing.
10. Janko Gravner and David Griffeath. Cellular automaton growth on z2: Theorems, examples, and problems. *Advances in Applied Mathematics*, 21(2):241 – 304, 1998.
11. Alexander E. Holroyd. Sharp metastability threshold for two-dimensional bootstrap percolation. *Probability Theory and Related Fields*, 125(2):195–224, 2003.
12. J. Kari. The Nilpotency Problem of One-dimensional Cellular Automata. *SIAM Journal on Computing*, 21:571–586, 1992.
13. Shiva Kintali and Sinziana Munteanu. Computing bounded path decompositions in logspace. In *Electron. Colloquium Comput. Complex.*, volume 19, page 126. Citeseer, 2012.
14. Nicolas Ollinger and Guillaume Theyssier. Freezing, Bounded-Change and Convergent Cellular Automata. *Discrete Mathematics & Theoretical Computer Science*, vol. 24, no. 1, January 2022.
15. Raphael M. Robinson. Undecidability and nonperiodicity for tilings of the plane. *Invent. Math.*, 12:177–209, 1971.
16. Klaus Sutner. Model checking one-dimensional cellular automata. *J. Cellular Automata*, 4(3):213–224, 2009.
17. S. M. Ulam. On some mathematical problems connected with patterns of growth of figures. In A. W. Bukrs, editor, *Essays on Cellular Automata*, pages 219–231. U. of Illinois Press, 1970.
18. Andrew Winslow. A brief tour of theoretical tile self-assembly. In *Cellular Automata and Discrete Complex Systems - 22nd IFIP WG 1.5 International Workshop, AUTOMATA 2016, Zurich, Switzerland, June 15-17, 2016, Proceedings*, pages 26–31, 2016.

A Proof of Theorem 1

Proof. Let $t \in \mathbb{N}$ a time, $\mathcal{A} = (G, \mathcal{F})$ a non-deterministic automata network and $\mathcal{E}_t$ a t-specification. First note that if G has bounded pathwidth, one can compute a path decomposition $\mathcal{P} = (X_1, \ldots, X_p)$ in **DLOGSPACE** where $p \leq \mathrm{pw}(G)$ [13]. Now note that we can adapt the previous **NC** algorithm to an **NL** algorithm in this particular context. First, observe that the dynamic programming lemma in [9] is also valid in this case, but now, because the decomposition is a path, there is only one bag for each level. Then, observe that testing whether a trace in compact representation (as explained earlier and presented in [9]) belongs to some regular language can be done in **DLOGSPACE**. Then, the algorithm will reproduce the same procedure than the algorithm in [9] , but, instead of parallelizing the information for the nodes in a bag storing it in different processors, it will handle this information non-deterministically. More precisely, an algorithm can guess a trace for each bag X_l from $l = 1$ to $l = p$ while ensuring that each node (that can appear in various bags) has the same trace in all guesses: this can be done because, by definition of a path decomposition, a node appears in an interval of $[1, p]$. This is the major difference with [9] that has to deal with tree decompositions. Thus, REGSPEC problem is in **NL**. □

B Proof of Corollary 1

Proof. For a freezing AN F over alphabet Q, the property of **not** being nilpotent is equivalent to the existence of a pair of orbits that ends up in two fixed points that differ at some node. For any pair of states q and q' and some node v, denote by $\mathsf{NONIL}(q, q', v)$ the problem of existence of two orbits in F that end respectively in states q and q' at node v. $\mathsf{NONIL}(q, q', v)$ is actually a REGSPEC problem for the AN $F \times F$ over alphabet $Q \times Q$ given by the following regular expression for trace at node v: $(Q \times Q)^*(q, q')^+$. Then, non-nilpotency can be expressed as the disjunction

$$\bigvee_{v \in V} \bigvee_{q \neq q'} \mathsf{NONIL}(q, q', v).$$

From this, we deduce a **NL** algorithm for non-nilpotency: choose non-deterministically one of the polynomially many instances of NONIL above and solve it in **NL** as an instance of the REGSPEC problem (Theorem 1). We deduce that nilpotency is **co-NL**. □

C Proof of Theorem 2

In order to show the main result, we need some technical definitions. Let Γ be a finite set and $\alpha \in \Gamma$. We call a string $y \in \Gamma^*$ an *α-marker* in $i \in [|y|]$ of length $l \in \mathbb{N}$ if $y|_{[i,i+l]} = \alpha \cdots \alpha$.

Given a directed graph (G, E) represented by its (oriented) incidence matrix M_G, we define for each $e = (u, v) \in E$ a block representing e as a $2 \times 2mn$ matrix $B(e)$ such that:

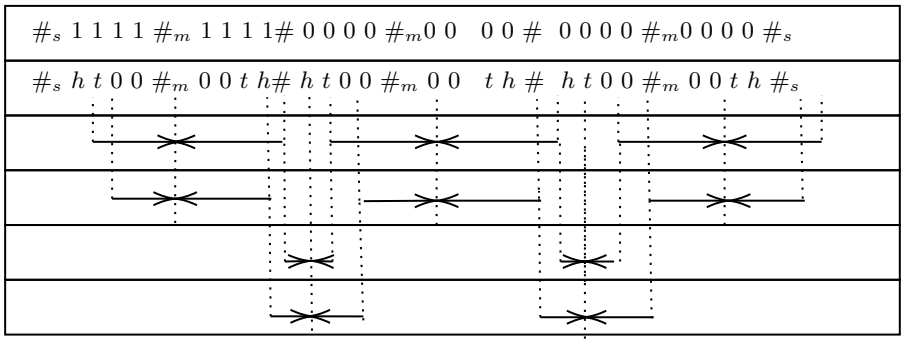


Fig. 2. Example of the verification dynamics for a periodic pattern. In this case, the pattern is given by the second row of a block representing the edge $(1, 2)$ in some graph.

1. $B(e)$ has $2m+1$ special symbols located at specific positions. More precisely $B(e)_{i,(n+1)(j-1)+1} \in \{\#_s, \#_m, \#\}$ for $i = 1, 2, 3$ and $j = 1, \ldots, 2m$.
2. Its first row $B(e)_{1\cdot} \in \{0,1\}^{(2n+2)m+1}$ is a 1-marker in $(2n+2)(e-1)+2$ of length n and a $1-$marker in $(3n+3)(e-1)+2$ of length n; and
3. $B(e)_{2\cdot} \in \{0,1\}^{(2n+2)m+1}$ is a periodic sequence of period $2n$ containing the row of M corresponding to e and a copy of this row in reverse order. More precisely, $B(e)_{2,[(2n+2)(i-1)+1,(2n+2)i]} = M_{\cdot,e}$ and $B(e)_{2,[(2n+2)i,(2n+2)(i+1)]} = \sigma(M_{\cdot,e})$ where σ is the permutation such that $\sigma(\{1, \ldots, k\}) = \{k, \ldots, 1\}$ for some fixed $k \in \mathbb{N}$.

For an example of a block for a some graph D see Figure 4.

First, observe that SPEC is in **NL** as a consequence of the Theorem 1. Now for the **NL**-hardness, let us take the problem STCON consisting in given a digraph $D = (N, A)$ and two nodes $s, t \in V$ deciding whether there exists a path connecting s and t. Let $(D = (N, A), s, t)$ be an instance of STCON. Observe that any path between s and t can be seen as a sequence of edges $e_1, \ldots, e_\ell$ such that $e_1 = (s, v)$, $e_i = (u', v') \implies e_{i+1} = (v', w')$ for some $u', v', w' \in N$, $i \in \{2, \ldots \ell - 1\}$ and $e_\ell = (w, t)$ for some $v, w \in N$. Besides, since each edge can be represented by a block then, an (s, t)-path P can be represented as a sequences of blocks $B(e_1), \ldots, B(e_\ell)$. Now, the main idea of the proof is to construct a non-deterministic automata network $\mathcal{A}_D = (G_D, \mathcal{F}_D)$ defined over a two dimensional grid of size $k \times d$ where $d = n^{\mathcal{O}(1)}$ and $k = \mathcal{O}(1)$. Of course, since k is constant, then $\mathcal{A}_D$ has bounded pathwidth. This automata network, will non-deterministically guess a sequences of blocks representing edges in A (selection phase). Needless to say that, the first part of the proof will be showing that $\mathcal{A}_D = (G_D, \mathcal{F}_D)$ is capable of deterministically verify if a n initial condition corresponds to a sequence of valid edges, i.e. if it corresponds to a sequence of blocks and they actually represent edges in A (verification phase). In order to perform these task, we use a construction based on using signals that will collide at specific locations as a way to verify the distance between two given cells. In addition, it would be essential to save (as a constant layer) the infor-

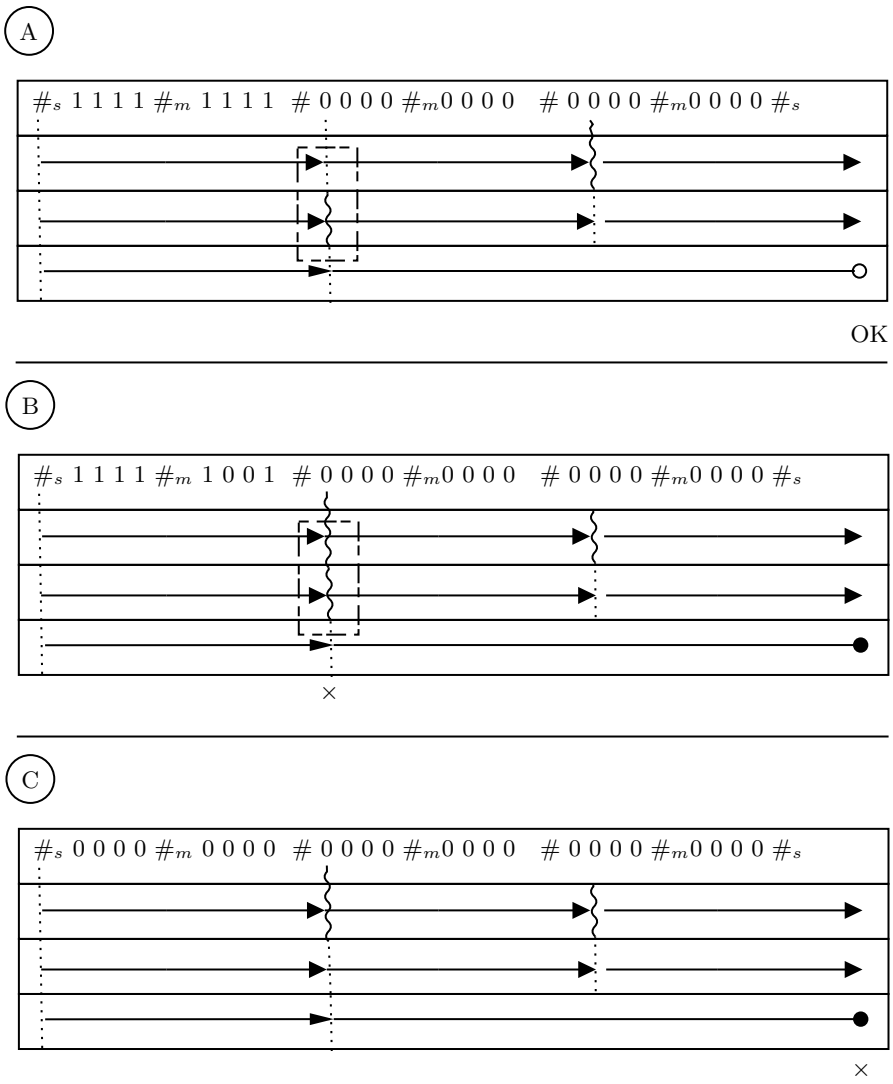
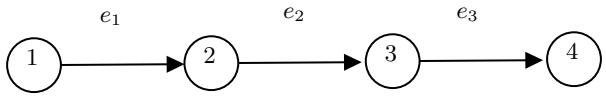


Fig. 3. Example of the verification dynamics for a marker. (Upper panel) A successful verification of a marker. If exactly one zone has only cells in state 1 an acceptance state will be reached. (Middle panel) An error in the verification raised by a zone in which cells in state 0 and 1 were identified. In this case, an error state is propagated. (Lower panel) An error in the verification raised by the signal only reading cells in state 0. In this case, an error state is propagated.

$B(e_1)$

$\#_s$ 1 1 1 1 $\#_m$ 1 1 1 1 # 0 0 0 0 $\#_m$0 0 0 0 # 0 0 0 0 $\#_m$ 0 0 0 0 $\#_s$
$\#_s$ h t 0 0 $\#_m$ 0 0 t h # h t 0 0 $\#_m$ 0 0 t h # h t 0 0 $\#_m$ 0 0 t h $\#_s$



D

Fig. 4. An example of a block for some graph D.

mation contained in the incidence matrix of D. Generally speaking, once $\mathcal{A}_D$ has verified that the sequence of blocks is valid, it will compare two subsequent blocks B_i and B_{i+1} in order to verify that if $B_i = (u, v)$ then $B_{i+1} = (v, w)$ for some $u, v, w \in N$. If in any part of its dynamics $\mathcal{A}_D$ detects some error, it will spread an error state that will lead the system to an attractor corresponding to a uniform configuration in which any cell will be in this particular error state. However, if the process runs flawless, then the system will reach an attractor in which all the cells are in a particular success state. We will code, by using a specification $\mathcal{E}_D$ (given in the input of SPEC), a specific requirement for the initial configuration (more precisely, we will ask the initial configuration to have the incidence matrix of D, markers and information about the nodes (s, t)) in order to allow $\mathcal{A}_D$ to have enough information to start the selection and verification process. In addition, we will code in this specification only the orbits that will reach this specific success state. By doing this, we will show that $\mathcal{A}_D \models \mathcal{E}_D$ if and only if there is a path between s, t in D. Thus, the reduction will consist on constructing $(\mathcal{A}_D, \mathcal{E}_D)$ from (D, s, t) in **DLOGSPACE**.

Now, we give details on the construction of $\mathcal{A}_D$ and $\mathcal{E}_D$:

1. $Q_D = Q_P \cup Q_{\text{core}} \cup Q_{\text{signal}}$ where $Q_{\text{core}} = \{\text{Success}, \text{Accept}, \text{Error}, 1, 0, \#_s, \#_m, \#\}$, $Q_P = \{P_1, \ldots, P_4\}$ are the states which indicate the different phases that are specified in the paragraph bellow and Q_{signal} are the states used in order to propagate signals (for example, the ones that we have used them on the previous lemmas).
2. Since we need to code an arbitrary path, d will be of size at most $m \times b$ where $b = (2n + 2)m + 1$ is the size of a block. Observe that size can be fixed since we can always assume that thexre is a loop in the terminal node t so we can consider that all the paths are coded by m blocks with possible padding of blocks $B((t, t))$. Thus, $d = n^{\mathcal{O}(1)}$.
3. k will be the number of rows of the grid. We will essentially use one row for the incidence matrix (incidence row), two rows for the blocks (selection row) and a constant number of rows that we will call working rows in which the signals will move and collide (working and verification rows).

4. $\mathcal{E}_D$ will code only initial conditions in which one row of the grid (incidence row) will have $\mathcal{O}(m)$ copies of the incidence matrix in the same format than the second row in blocks i.e. there are markers at specific positions and we code the different columns in the zones defined by the markers (Figure 5).
5. $\mathcal{E}_D$ will code orbits in which the selection row of size d will have marked in the first block a symbol indicating "head" in the position associated to node s (see Figure 4).
6. $\mathcal{E}_D$ will code orbits in which the selection row of size d will have marked in the last block a symbol indicating "tail" in the position associated to node t (see Figure 4).
7. $\mathcal{E}_D$ will code orbits which will reach a uniform success state.

Now we will describe the dynamics of the automata network $\mathcal{A}_D$.

Initialization In $t = 0$, since by construction of the specification $\mathcal{E}_D$, we can consider only orbits in which the incidence row, all the special symbols and the position of the source and terminal node (as an h and t symbol fixed in its correspondent positions) are well coded and fixed in the initial condition.

Selection phase First, the local rules will non-deterministically guess the states of the rest of the cells in the selection row. This process is performed cell by cell, by sending a traveling signal in one of the working rows. This signal starts on a starting symbol $\#_{\mathrm{s}}$ and finishes in a terminal symbol $\#_{\mathrm{s}}$. After doing that, the signal comes back from the terminal symbol to the starting symbol and writes a change of phase state in all the cells on the working row.

Verification phase After that, verification phase starts. The process has two main subphases:

A local phase: First, each block is internally verified. More precisely, the local rules will verify that $B(e)$ has the correct formatting on its two rows and that it correspond to an actual edge in D, i.e. $e \in A$:

1. *Verification of the first row.* For each part of size $2n$ defined by two different special symbols (i.e. the space bounded by pairs $(\#_s, \#), (\#, \#_s)$ or $(\#, \#)$)), three different signals will start from one symbol to the one in its left (see Figure 3). The first signal will change of state if and only if it reads a cell in state 0. If it remains in initial state it will be interpreted as success otherwise, if it has changed, then it will be interpreted as error. The second one will do the same thing but for cells in state 0. Finally, the third signal will start from a cell marked with $\#_s$ and will go through the row until another $\#_s$ symbol is reached. This signal will verify that there is exactly one block which has marked success for the first signal and error for the second one. Otherwise, it will change to an error state that will be spread to all the cells (see Figure 3 for examples).
2. *Verification of the second row.* In order to verify that the coding of the second row is correct, we need to check that each row has exactly two symbols: h

and t and that the configuration is symmetric related to the cells marked with symbols $\#_m$. In order to do that, from each symbol h and t a signal is sent through two working rows (one signal to the right and one to the left, see Figure 2 for details.) Then, the local rules in the cells holding the state $\#_m$ will change to the success if exactly two signals arrive at the same time. More precisely, this last procedure is implemented by sending a two state signal, one marking the starting part of the signal and one marking the rest. If the latter condition does not hold, the cells marked by $\#_m$ will spread an error state (see Figure 2). Observe that this procedure works since: i) if two cells are holding the same state and they are at the same distance of the cell marked by $\#_m$ then, the two equal signals will arrive at the same time to the cell holding the sate $\#_m$; and ii) since the coding considers a constant amount of special symbols (more precisely h and t) then, the local rule is freezing.

3. *Verification of the edge that is coded in the block.* At this point, if no error state has been produced by the dynamics, it means that the coding of each block is coherent, but we are not sure that it actually represents an edge $e \in A$. In fact, we have coded in the first selection row some number i referencing a column of the adjacency matrix of D but, we need to check whether the second selection row contains the same information than the i-column of the adjacency matrix. This last part is performed in the following way: a signal will be transmitted over a working row in order to identify the information in the two selecction rows of the block. Since each block has a marker in its first row, the signal can hold a state while it is in the same position than the cells in state one in the marker. Thus, this state will indicate the local rule to perform a comparison between the second row of the block and the correspondent part of the incidence row. For more details see Figure 5. While verifications are being run, the local rule will write an acceptance state or an error state in some working row. Finally, a third signal will verify that all the cells in the latter working row are in the acceptance state and will spread the error state if not. Finally, if no error state has been spread, the local rule updates the state of the cells in the working row holding the change of phase state.

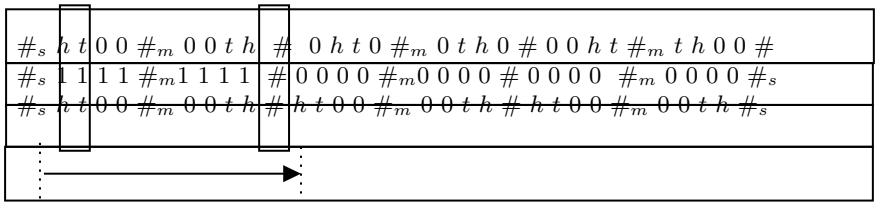


Fig. 5. Example of adjacency verification. In the first row, the incidence matrix of D is coded. In this case a signal verifies that the edge $(1, 2)$ is in the graph D (see Figure 4)

A pair-wise coherent phase: similarly to the verification of the second row in the selection row on the previous phase, this phase starts by sending multiple signals that are sent from the cells in the selection row with states given by the symbols marking the tails and the heads of the coded edge in the second row of the selection row on each block. These signals are sent through two different working tapes. Each of these signals will carry a special state indicating if its origin was a head or a tail. The local rule in the cells with a special symbol ($\#_s$) will verify whether a head signal has collided with a tail signal (see Figure 6). If exactly one of this collision take place, the local rule will write an accept state in one of the working rows. Finally, in other working row, a signal starting from the starting symbol will verify that at the position of the beginning (ending) of a block an accept state is written in the previous working row. The local rule will update the cells in that working row to an error state that will spread if at least one the verifications is not correct. Otherwise, it will update the cells to the success state.

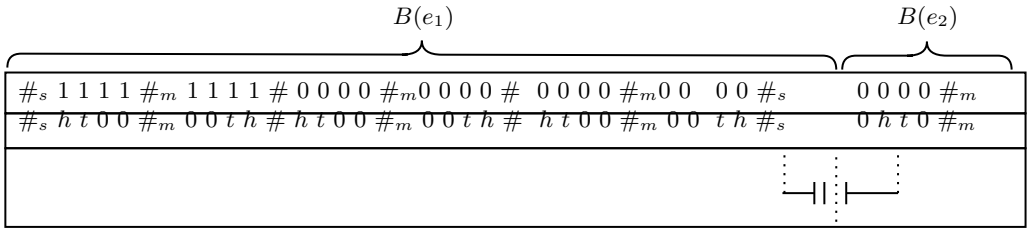


Fig. 6. Example of a verification dynamics which compares blocks and checks if the corresponding edges are both incident to the same node. If exactly one tail signal collide with a head it means that the previous property is verified and an error state is spread otherwise.

Now, we turn into show that the reduction hold. First, observe that the construction of $(\mathcal{A}_D, \mathcal{E}_D)$ can be done in **DLOGSPACE** since local rules does not depend on the structure of D and thus, we only need to store partial information related to the structure of the incidence matrix of D in order to define the specification. Then, we have that if there is a path between s and t on D, by construction, there must be at least one orbit of $\mathcal{A}_D$ which satisfies $\mathcal{E}_D$. Conversely, if $\mathcal{A}_D \models \mathcal{E}_D$ then, there exist at least one initial condition which codes a sequence of edges in D which leads the system to a uniform success fixed point. By construction, this attractor is only reachable (starting from the set of valid initial conditions) after all the previous phases are successfully performed by the dynamics. Then, we deduce that $\mathsf{STCON} \leq_m^{\mathbf{DLOGSPACE}} \mathsf{SPEC}$ and thus, SPEC is **NL**-hard. The theorem holds.

D Proof of Lemma 1

Proof. There exists a Turing machine working in polynomial time (and space) that on input (Ψ, v) where Ψ is a SAT formula and v a candidate valuation checks whether v satisfies Ψ. Then for any given SAT formula Ψ, one can produce in LOGSPACE a set of HV-domino constraints that accepts only bi-dimensional configurations $(a_{i,j})$ which represent a valid space-time diagram of the above machine which are correctly initialized and with Ψ enforced as the first component of the input. The encoding of space-time diagram of Turing machine inside domino constraints is well-known and usually presented through a fixed set of so-called Wang tiles (see for example [15]), which are just a uniform set of horizontal constraints $H \subseteq Q^2$ and vertical constraints $V \subseteq Q^2$. Note that since the HV-domino constraints considered here are non-uniform, we can hard-code the initial state of the machine in the lower-left corner of the configuration, the encoding of Ψ in the initial row, and the accepting state of the machine in the top row. The reduction from SAT to the HV-domino CSP follows. □

E Proof of Theorem 3

Proof. We proceed by reduction from the HV-domino CSP: given n and constraints $(H_{i,j})$ and $(V_{i,j})$, we build a deterministic automata network $(G_N, \mathcal{F})$ with $N = n^2$ which verifies ϕ if and only if the CSP has a solution. G_N is the graph with nodes $V = \{1, \dots, N\}$ and edges (i, i) for all $i \in V$ and $(i, i+1)$ for all $i < N$ and $(i, i-1)$ for all $i > 0$. G_N has pathwidth 1 and degree 3. The automata network $\mathcal{F}$ has four components plus a global error state and uses alphabet $Q' = Q \times \{0,1\} \times Q_h \times Q_t \cup \{\bot\}$ (where Q is the alphabet of the HV-domino CSP). The freezing order on $Q \times \{0,1\} \times Q_h \times Q_t$ is simply the product of orders on each component, and this order is extended to Q' by taking $\bot$ as a maximal element. The overall behavior is as follows (see Figure 7 and Figure 8).

- $\bot$ is an invariable spreading error state: as soon as some node is in state $\bot$, its neighbors change to $\bot$ in one step.
- The Q-component contains a candidate configuration $(a_{i,j})$ written as a one-dimensional word $a_{1,1} \cdots a_{1,n} a_{2,1} \cdots a_{2,n} \cdots a_{n,1} \cdots a_{n,n}$. The block of nodes $(j-1)n+1$ to $(j-1)n+n$ will be referred to as *block* j and it contains line j of the matrix $(a_{i,j})$ in its Q-component. This component never changes, except when an error state $\bot$ invades the network, or when some H-constraint $H_{i,j}$ is violated at some node in which case a $\bot$ state is generated. The freezing order on this component can be chosen arbitrarily.
- The $\{0,1\}$-component is called *dummy component* which never changes, has no influence on other components, and is just here to ensure that any configuration leading to $\bot^N$ has enough preimages (see Claim E below).
- The Q_h component handles a global control *head* whose main behavior is a back-and-forth movement from node 1 to node N and back to node 1. More precisely, the head do so on a set Σ of well-formed configurations and

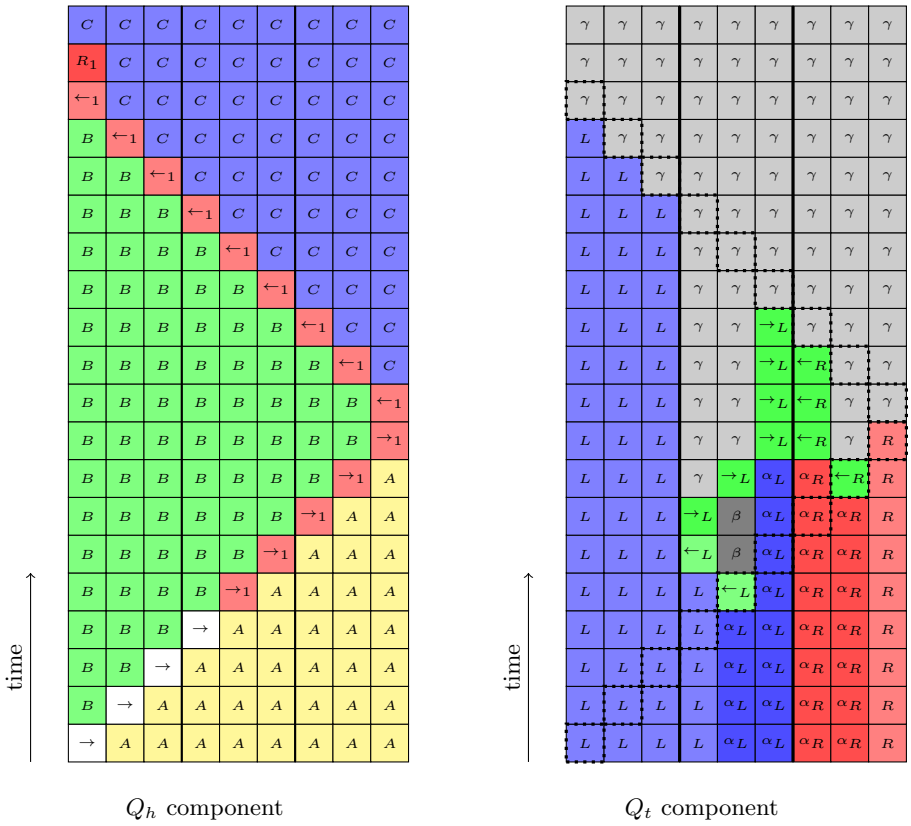


Fig. 7. Example of valid orbit with $n = 3$ starting from a configuration testing the V-constraint $V_{2,2}$ and resulting in a positive output. The trajectory of the Q_h head is reproduced on the Q_t-component to clarify the interactions. The vertical thick lines represent separations between consecutive blocks.

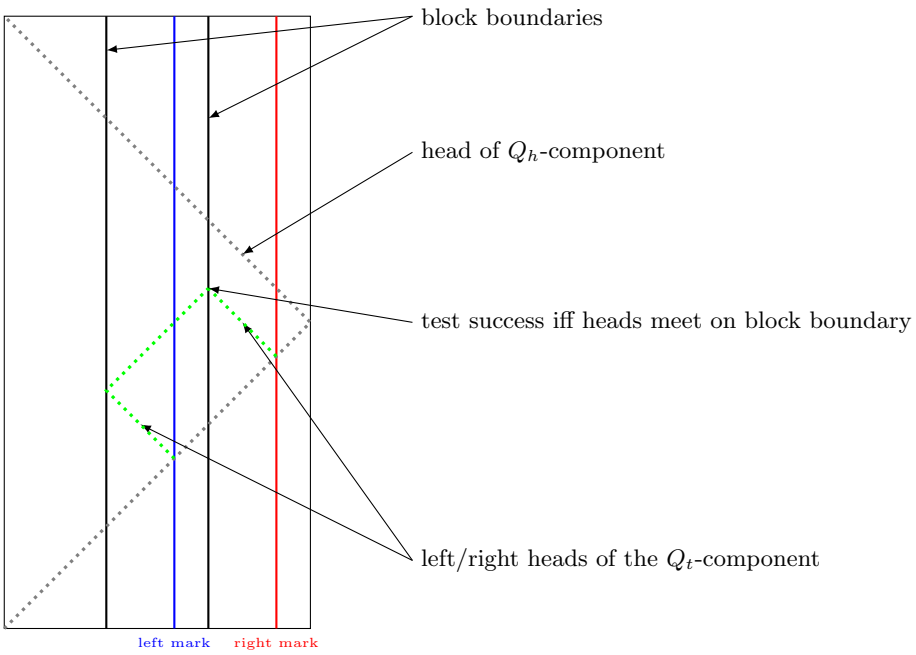


Fig. 8. Euclidean non-discretized representation of the verification process ensuring that the marks in two consecutive blocks have the same offset within the block and thus mark two positions which are vertical neighbors in the matrix $(a_{i,j})$, *i.e.* of the form $a_{i,j}$ and $a_{i+1,j}$.

any ill-formed configuration is detected locally and generates an error state $\perp$. We set $Q_h = \{0,1\} \times \{0,1\} \times \{A, B, C, \rightarrow, \rightarrow_0, \rightarrow_1, \leftarrow_1, \leftarrow_0, R_0, R_1\}$ with freezing order $0 < 1$ on the two $\{0,1\}$ components and $A <\rightarrow<\rightarrow_0<\rightarrow_1< B <\leftarrow_1<\leftarrow_0< R_0 < R_1 < C$ on the remaining component. Σ is defined by forbidding a set of pairs of states to occur two adjacent symbols $c_i c_{i+1}$ from the third component. Moreover, we add the constraint that node 1 cannot be in state A, and that the first $\{0,1\}$ component at this node must be 1. Precisely, configurations authorized in Σ are the following (without considering the $\{0,1\}$ components):

1. $\rightarrow A^{N-1}$ or $\rightarrow_0 A^{N-1}$ or $\rightarrow_1 A^{N-1}$,
2. $B^i \rightarrow A^{N-i-1}$ or $B^i \rightarrow_0 A^{N-i-1}$ or $B^i \rightarrow_1 A^{N-i-1}$,
3. $B^{N-1} \rightarrow_0$ or $B^{N-1} \rightarrow_1$,
4. $B^{N-1} \leftarrow_0$ or $B^{N-1} \leftarrow_1$,
5. $B^i \leftarrow_0 C^{N-i-1}$ or $B^i \leftarrow_1 C^{N-i-1}$,
6. $\leftarrow_1 C^{N-1}$ or $\leftarrow_0 C^{N-1}$,
7. $R_1 C^{N-1}$ or $R_0 C^{N-1}$,
8. C^N.

The head is the unique arrow occurring in each configuration and its dynamics is as follows. It moves to the right in a background of As and letting symbols B behind (configuration types 1, 2 and 3). At each move to the right, the first $\{0,1\}$ component of the node left by the head is reset to 1. When doing so it can turn at some point to state $\rightarrow_1$ or $\rightarrow_0$ depending on the layer of states Q_t as detailed below: these states represent a head holding a YES/NO bit of information about the output of the test process happening on component Q_t. This bit must appear before reaching node N and once appeared, this bit of information never changes in the future. When reaching node N the head starts to move to the left, progressing in a background of Bs and letting symbols C behind (configuration types 4 and 5). At each move to the left, the second $\{0,1\}$ component of the node left by the head is reset to 1. The fact that some $\{0,1\}$ component is reset to 1 at each head move ensures that the corresponding configurations have more than one preimage (which is a key aspect when considering formula ϕ). Finally, the head reaches node 1 and must hold the output bit of the test process (configuration type 6), maintain it one step (configuration type 7), and finally erase it (type 8). Also, when reaching a configuration of type 6 at node 1, the bit of the second $\{0,1\}$ component is reset to 1 when the head at node 1 is $\leftarrow_1$ and unchanged when it is $\leftarrow_0$. This bit is reset to 1 in any case for configurations of type 7. As a result, a configuration of type 7 has exactly one preimage if and only if it is $R_1 C^{N-1}$.

– The Q_t component is the *test* component, its role is to mark two positions in the configuration and interact with the head component in order to check a single V-constraint on the candidate configuration hold in the Q-component. More precisely, the test component ensures that the two marked positions are at distance n (*i.e.* they correspond to two vertical neighbors in the grid $(a_{i,j})$) and gathers locally at some node the information on the corresponding pair of states $a_{i,j}a_{i,j+1}$ and the constraint $V_{i,j}$ so that the head can check

whether $a_{i,j}a_{i,j+1} \in V_{i,j}$. See Figure 8 for an Euclidean intuition of how the distance equality test works. This behavior is implemented using alphabet $Q_t = Q \times Q \times \{L, R, \alpha_L, \alpha_R, \beta, \gamma, \leftarrow_L, \leftarrow_R, \rightarrow_R\}$ with freezing order: $L < R < \alpha_L < \alpha_R <\leftarrow_L< \beta <\rightarrow_L<\leftarrow_R< \gamma$. The third sub-component of Q_t is used to mark two positions in the configuration as well as check that the distance between the two marked positions is exactly n so that they indeed correspond to a pair of positions (i, j) and $(i, j + 1)$ in in the matrix $(a_{i,j})$. Its behavior is based on a set of valid configurations Σ^+ defined by local rules and synchronized with the Q_h component. States $\leftarrow_L, \leftarrow_R, \rightarrow_R$ are called "left/right arrows" of the Q_t-component and are generated at specific positions when the global Q_h-head passes by (see Figure 8). The two Q-sub-components of Q_t are forced to hold states $a_{i,j}$ and $a_{i,j+1}$ respectively on valid configurations, and allow to check the V-constraint $(a_{i,j}, a_{i,j+1}) \in V_{i,j}$. The conditions defining Σ^+ are local (*i.e.* they can be defined as a list of admissible pair of states between neighboring nodes) and a $\perp$ state is triggered whenever and wherever an invalid local pattern is detected. The conditions are the following:

- First, in the absence of a left-moving head in the Q_h component, the two Q-sub-component must be uniform: each one is of the form q^N for some $q \in Q$. When there is a left-moving head in the Q_h component, each Q-sub-component is of the form: $q^i q_0^{N-i}$ where q_0 is the maximal state of Q and i is the position of the Q_h head.
- Then, there are five types of admissible configurations on the third sub-component of Q_t:
  1. $L^*\alpha_L^+\alpha_R^+R^*$ and α_L and α_R segments are forbidden to cross a block boundary (*i.e.* node (n, j) has an α_L if and only if $(1, j + 1)$ has an α_R),
  2. $L^* \leftarrow_L \beta^*\alpha_L^*\alpha_R^*R^*$,
  3. $L^*\gamma^* \rightarrow_L \beta^*\alpha_L^*\alpha_R^*R^*$ or $L^*\gamma^* \rightarrow_L \beta^*\alpha_L^*\alpha_R^* \leftarrow_R \gamma^*R^*$,
  4. $L^*\gamma^* \rightarrow_L\leftarrow_R \gamma^*R^*$,
  5. any configuration of the form $c\gamma^*$ where c is the prefix of a configuration of type 3 or 4.
- Type 4 configurations are only authorized when $\rightarrow_L$ and $\leftarrow_R$ states meet at a bloc boundary, *i.e.* are at positions of the form (n, j) and $(1, j + 1)$ (respectively).
- Moreover, only L, α_L, α_R and R are authorized in a node whose Q_h component is in state A, therefore a type 1 configuration on the Q_h component admits only a type 1 configuration on the Q_t component.
- Finally, in configurations of type 1, let (i, j) (*i.e.* $n(j - 1) + i$) be the leftmost node in state α_L and let m be the rightmost node in state α_R. Denote by a and b the states of the first and second Q-sub-component respectively. Then it must hold that a is the state of the Q-component (the global one of the alphabet Q') of node (i, j) and b is the state of the Q-component of node m.

The dynamics of this Q_t-component is as follows and respects the type order of configuration described above:

- Type 1 configurations don't change until the head of the Q_h-component arrives at node (i, j) where it generates a $\leftarrow_L$ state.
- Then, $\leftarrow_L$ propagates in the L background, letting β states behind and until position $(1, j)$ is reached (*i.e.* the first position to the left which is at the beginning of a bloc). Then, the arrow bounces by turning into $\rightarrow_L$ and starts to progress to the right letting γ states behind.
- Meanwhile, when the Q_h head reaches position m (the rightmost node in state α), it launches a $\leftarrow_R$ state in the Q_t layer which starts to propagate to the left letting γ sates behind.
- Also, when the Q_h head bounces on node N and starts to propagate to the left, it writes q_0 on each Q-sub-component and γ on the third sub-component of Q_t, thus erasing progressively any information about the marked positions and the V-constraint being tested.
- The dynamics ends into the fixed point equal to q_0^N on each Q-sub-component and γ^N on the third sub-component.

Finally the Q_t-component influences the Q_h-component as follows: when the head of the Q_h-component of type $\rightarrow$ reaches node (i, j) it becomes $\rightarrow_1$ if $(a, b) \in V_{i,j}$ (where a and b are the states of the Q-sub-components) and $\rightarrow_0$ else.

Let us now prove that this construction has the desired property. Let's call valid orbit any orbit without occurrence of $\perp$.

Claim (ϕ checks V-constraints on valid orbits). Consider any valid orbit starting from a configuration y without preimages, with $y \rightarrow y^1$ and $\neg P_2(y^1)$, and reaching a fixed point x. Then y is of type 1 on components Q_h and Q_t. Moreover, a correctly encoded test of V-constraint $V_{i,j}$ is encoded in component Q_t and the configuration z such that $y \rightarrow^+ z$ and $z \rightarrow x$ verifies $\neg P_2(z)$ if and only if $a_{i,j}a_{i,j+1} \in V_{i,j}$.

Proof. Since there is no occurrence of $\perp$, the whole orbit belongs to Σ^+. A configuration of type 8 or 9 in the Q_h component always has a preimage so y is not of this type. A configuration of type 2,3,4,5,6 or 7 has a moving head that reset some $\{0, 1\}$ component to 1, so it cannot be the unique preimage of its successor, contradicting the hypothesis on y^1. Therefore y is of type 1 on components Q_h and Q_t. Then, by construction, the marked positions in the Q_t component are at distance n and there is a well-formed V-constraint test happening (otherwise a $\perp$ would be generated later in the orbit). The dynamics of the automata networks then ensures that the Q_h heads holds the bit of information corresponding to the validity of the encoded V-constraint: it is 1 if and only if $a_{i,j}a_{i,j+1} \in V_{i,j}$. The dynamics ends in a fixed point x which has a configuration of type 8 on the Q_h-component. Already when reaching a configuration of type 6 or 7 or 8 on the Q_h-component, all the Q_t-component has been reset to a default value. Therefore it holds that the bit of information in the head is 1 if and only if the type 7 configuration reached $z = R_1C^{N-1}$ has a unique preimage. □

From the construction and Claim E it should be clear that if the HV-domino CSP has a solution $(a_{i,j})$, then one can encode it into a fixed point configuration

x that satisfies the orbit property expressed in ϕ for all admissible choices of initial configuration y (because all admissible V-constraint tests are satisfied by the CSP solution). In this case the automata network verifies ϕ.

Conversely, if the automata network verifies ϕ and if the fixed point x can be chosen to be a configuration without $\perp$, then this configuration encodes a solution to the HV-domino CSP by Claim E and because any valid V-constraint test can be encoded in an appropriate initial configuration y. It remains to discard the possibility that ϕ is valid because x is chosen to be the invalid fixed-point $\perp^N$, this is the purpose of the following claim.

Claim (ϕ discards invalid orbits). Consider three configurations y, z, x such that $y \to^+ z \to x$ and $x \to x$ and $x \neq z$. If $\neg P_2(z)$ then x cannot be the configuration $\perp^N$.

Proof. First z must have an occurrence of $\perp$ because it is impossible that the preimage z' of z be everywhere correct and in one step becomes a configuration z everywhere incorrect but without occurrence of $\perp$: indeed, by construction, the changes not involving $\perp$ state that can occur in a configuration in one step are only in the neighborhood of arrow states of both Q_h and Q_t components, and they have a bounded number of occurrences by definition of Σ^+. Moreover, there must be an occurrence of $\perp$ in z at position i such that $z'(i) \neq \perp$. Indeed, otherwise it would imply $z = \perp^N$ which is impossible under the hypothesis. Therefore by just changing the dummy component at i in z' we produce another preimage of z, so $P_2(z)$ holds which is a contradiction. □

We have thus shown that the HV-domino CSP has a solution if and only if the automata network verifies ϕ. The theorem follows since the construction can be computed efficiently (actually in LOGSPACE). □